Securing the Cloud

By Tanmay Kaushik

Student Number - 18308341

INTRODUCTION

In today's time, the need of a secure cloud storage application is much needed. Specially for an organisation or a team which uses applications like Google Drive, Dropbox, Office365 to store and exchange documents with each other.

I was asked to implement a solution where a group of users can securely upload and exchange files through cloud storage application using key management, public-key certificates and control management.

To achieve this, I use multiple libraries and tools for securing the Google Drive storage. I use the Drive API provided by Google to download and upload files.

Then, to control the access to the files, I create a file which is used to Add or Remove users from the group who have the access to that files. The file which stores the users credentials is a protected using encryption.

Along with access management, I add AES encryption to encrypt the files which will be uploaded on the Google Drive. I create an AES key which will be used for encryption and decryption.

To share the keys, I use RSA encryption which employs public-key certificates. RSA encryption is used to create Public and Private Keys for each user which will be used to share the AES key which is needed to access the encrypted files on Google Drive.

To keep the group protected, I have created a role of an Admin. Admin is solely responsible for Adding and Removing people from the group. When the Admin creates a user, Public and Private Keys for the user are automatically generated.

REQUIREMENTS

Key-Management System

The project requires us to use Key-Management to protect the files which will be uploaded on the cloud storage application. To implement this I use AES-128 key-management system with the help of Python's PyCryptodome library. I generate a random key of length 16 bytes and store in a file called *"key.key"*. This key will be used to encrypt a file when uploading and decrypt a file when it is downloaded.

Along with the key, I add an Initialisation Vector when the file is encrypted to create random blocks which is essential when we are dealing with block encryption. For each file, a new and random IV is generated to ensure that each encrypted block is unique.

Share Files Securely

One of the requirements of the project is to create a solution where the user can share the files securely. I have developed a solution which uses Google Drive to share files. An user can upload and download files from the google drive only if the user is a part of the group. Along with the access management, I use AES encryption.

When the user selects a file to upload on the Google Drive, the file is firstly encrypted using the key (which is stored in the *"key.key"* file) and a random Initialisation Vector. The encrypted file is stored in the Uploads Folder. In case the file cannot be encrypted for some reason, an option is given to the user if he/she wants to upload the file without encryption or not.

After the encryption stage, the program uploads the encrypted file from the Uploads Folder to Google Drive. If the file is uploaded correctly, the encrypted file is removed from the local storage.

When the user wants to download a file from the Google Drive, the program accesses the Drive API and invokes the download function. Then the downloaded file that is checked if the file requires decryption or not since there might be some files on the Drive which were never decrypted. If the file is found to be encrypted, the file is decrypted using the key which is available to the user. The decrypted file is stored in the Downloads folder and the encrypted file is deleted.

In case the user does not have the access to the AES key for encryption and decryption, admin can run the "rsa_encryption.py" file and obtain the public key of the user. Then the admin can use that public key to share the AES key with the user which is used for encrypting and decrypting files on Google Drive.

Public-Key Certificates

The project requires us to use Public-Key certificates which enables users to share files securely. As the solution that I have created uses AES key for encryption of files uploaded on Google Drive, I created a solution using RSA to share AES key in case any user looses their personal key.

Each user which is registered in the secure group has their own combination of Public and Private keys. I created the solution in such a way that when the Admin adds a new user to the secure group, a sub-directory is created in the Users Folder. This sub-directory contains the public and private keys of the users. Thus, when the user requires a new copy of the AES key, the Admin can simply access the Public key of the user, encrypt the AES key file with the Public key and share the encrypted file with the user.

The encrypted AES key can be decrypted by the user who has their private key which was generated when the new user was set-up. This is how I use public-key certificates using RSA to share files securely.

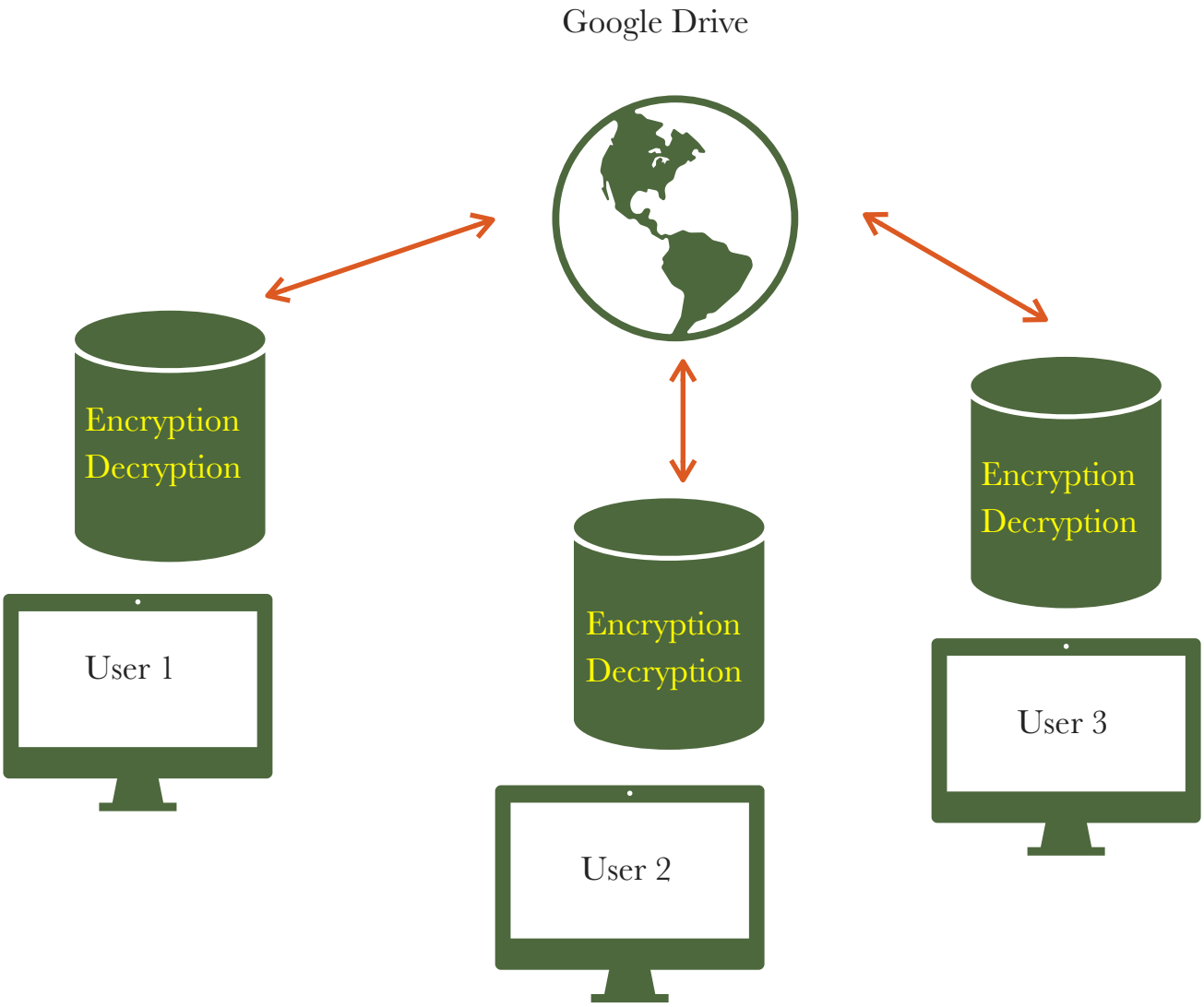
Add or Remove Users

One of the major requirement of the project is the functionality to add and remove users. I was asked to implement a solution where only one person is responsible for access control. So I created a role of Admin (where username is Admin and Password is known only to the admin). Only Admin has the authority to add or remove users. The Admin has a special menu which has functions like Add or Remove users.

When the Admin Adds a new user, the Admin is asked the username and password for the new user. When Admin signs up the new user, the Public and Private keys for the user are generated and stored in the Users folder.

DESIGN OVERVIEW

The purpose of this project is to implement a solution where a group of users can share files securely. I have explained few scenarios below which shows how the solution I implanted works:



The Admin is responsible for adding and removing users which will have access to Google Drive. Users in the group created by the admin are the only people who can upload/ download files.

If User 1 wants to upload a file to Google Drive, the user selects the file from his/her local computer and puts it into the Uploads Folder. Then the file selected by the user will be encrypted using the key which is stored on the user's computer. This is a shared key which is available to every users of the group. After the file is encrypted using AES-128 key management, the encrypted file is uploaded on the Google Drive. This adds an extra level of

security which restricts people who are not in the group to read the file even if they manage to get the file from somewhere as they do not have access to the key.

If User 2 wants to download the file which was uploaded by User 1, the user selects the filename and enters it in the program. Then the file is downloaded in the Downloads folder of User 2's local machine. Then the file is checked if it is encrypted or not. Since the file uploaded was encrypted by User 1, the file needs to be decrypted. Encrypted file is decrypted using the AES key which is available to all the users in the group. The decrypted file is now stored in the downloads folder and the encrypted file is deleted.

Now if User 3 wants to download the file which was uploaded by User 1 but User 3 has lost or deleted his/her AES key. This will require User 3 to get the key from the Admin. As Admin has to send the AES key to User 3, the AES key file needs to be encrypted to keep the key protected from outside world. This is done using RSA Public and Private Keys. Admin encrypts the AES key file with User 3's Public Key and sends it to the user. Either the Admin can send it via Email or Upload it on the Drive. Since the file is encrypted it can only be accessed by User 3. User 3 can download the encrypted key from the Drive and decrypt it using his/her own Private Key to access the AES key. Not User 3 has the AES key which can be used to decrypt User 1's file.

IMPLEMENTATION and WORKING

To run the program, you will need to download a file called client_secret.json from the Google Drive API website. This file is required to establish an authenticated exchange.

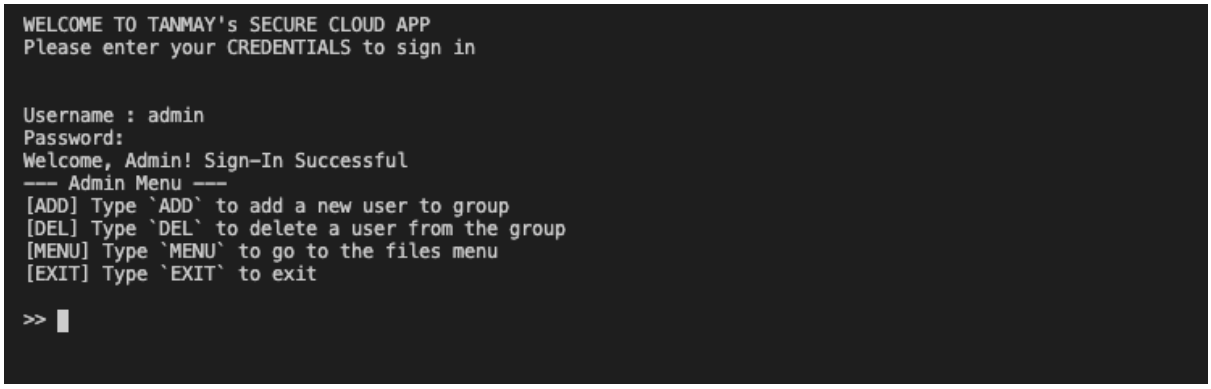
The steps to download the file and enable the Google API can be found here :
`https://developers.google.com/drive/api/v3/quickstart/python`

After the Google Drive API is enabled and the file is downloaded, one can run the main.py file. The program might ask one to authenticate the new computer which is a one time process. A link will be provided to the user where the user can navigate to allow the use of this application. Once this authentication is finished, the program can function properly without any hindrance from Google's extra security.

ADMIN MENU

When the user will run the program, the program checks the file which stores the credentials of all the members of the group. If the file is empty (which means there are no members), the user will be prompted to Create an Admin account.

When the Account is set up, the admin I can login as Admin where he/she will be shown a list of options :

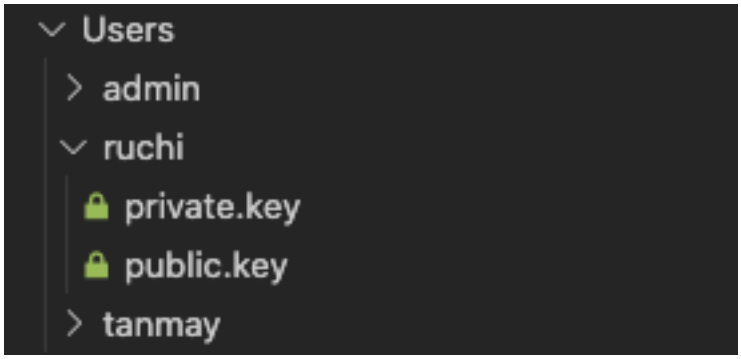


This Menu is only available to the Admin of the group as only the Admin can Add or Delete users to the group.

Admin can Add a new user -

1. When Admin tries to Add an User by the name of "Tanmay", the program prompts to select a new name since it was already taken.
2. Then the Admin selects the name "Ruchi" and the account for Ruchi is created.
3. When Sign-Up is successful, Public and Private keys for Ruchi are generated.

```
>> ADD
--- SIGN-UP ---

Please choose an Username & Password to Sign-Up

To go back, enter `BACK`
Username : Tanmay
This username has been taken. Please Choose a new one.
Username : Ruchi
Password:
.
.
Sign-Up Successful.

Generating Public and Private Keys for  ruchi
Public and Private Keys of the user created in the Users folder
```



Public and Private Keys created for Ruchi in the Users folder.

Similarly, Admin can Delete users from the group which will remove the user's credentials from the file that stores everyone's user credentials. This will revoke user's access to Google Drive.

When anyone other than Admin logs in to the application, they will be shown a list of options which includes Uploading, Downloading and Viewing files from Google Drive.

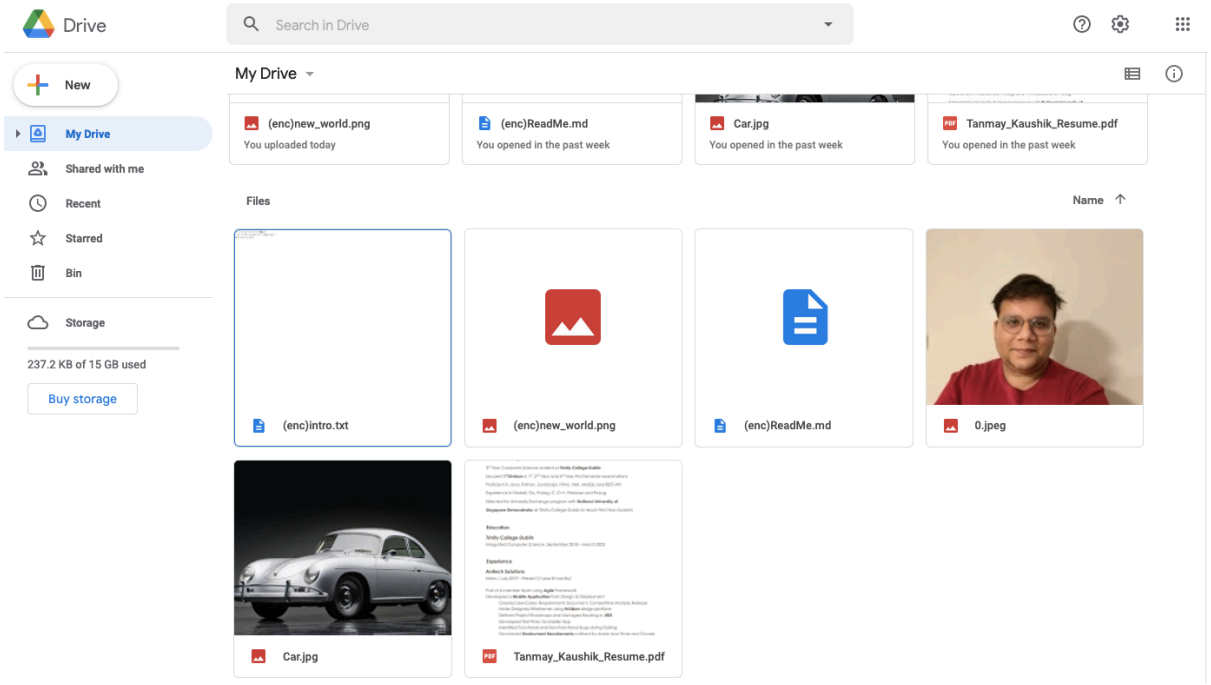
```
WELCOME TO TANMAY's SECURE CLOUD APP
Please enter your CREDENTIALS to sign in


Username : Tanmay
Password:
Sign-In Successful

--- Menu ---
[UPLOAD] Type `UPLOAD` to encrypt a file and upload it on Google Drive
[DOWNLOAD] Type `DOWNLOAD` to download a file and decrypt it
[LIST] Type `LIST` to list the files on Drive
[EXIT] Type `EXIT` to exit

>>
```

The currently logged in user is “Tanmay” and the options available to the user are shown below:

View Files on Google Drive

When the user types in “LIST”, the files currently on Google Drive are shown. Files with “(enc)” in front of their names indicate that those files are Encrypted.

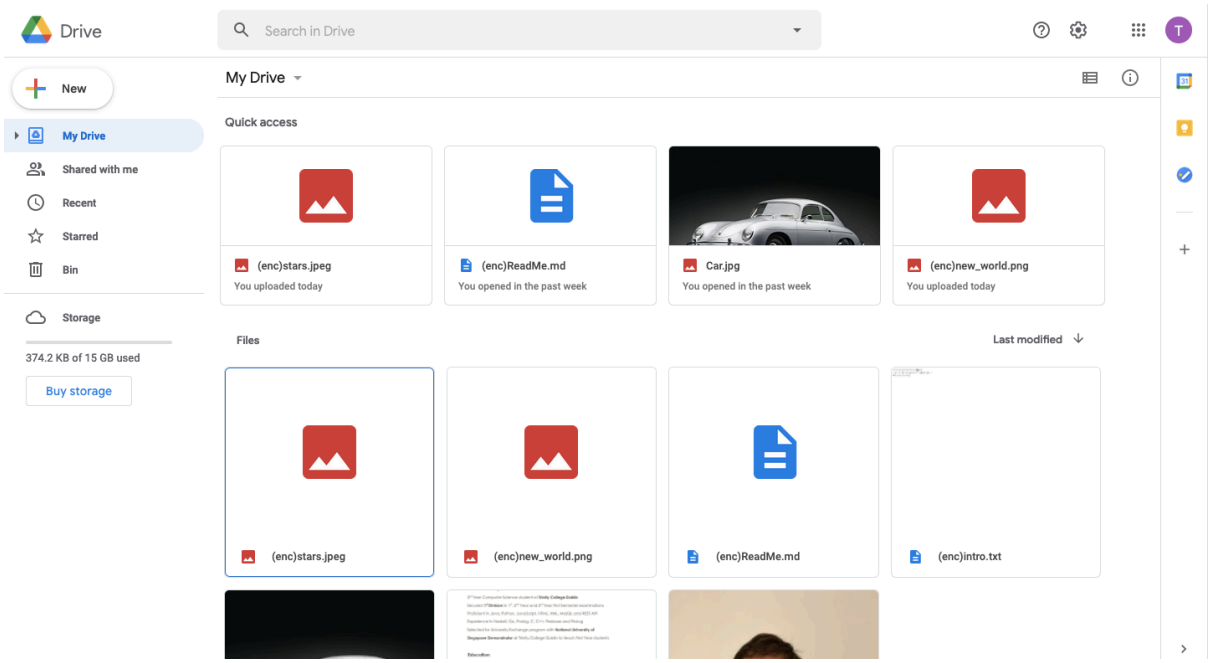
```
>> LIST


Files on Google Drive:

(1)(enc)new_world.png
(2)(enc)ReadMe.md
(3)(enc)intro.txt
(4)Car.jpg
(5)Tanmay_Kaushik_Resume.pdf
(6)0.jpeg
```



Upload Files on Google Drive

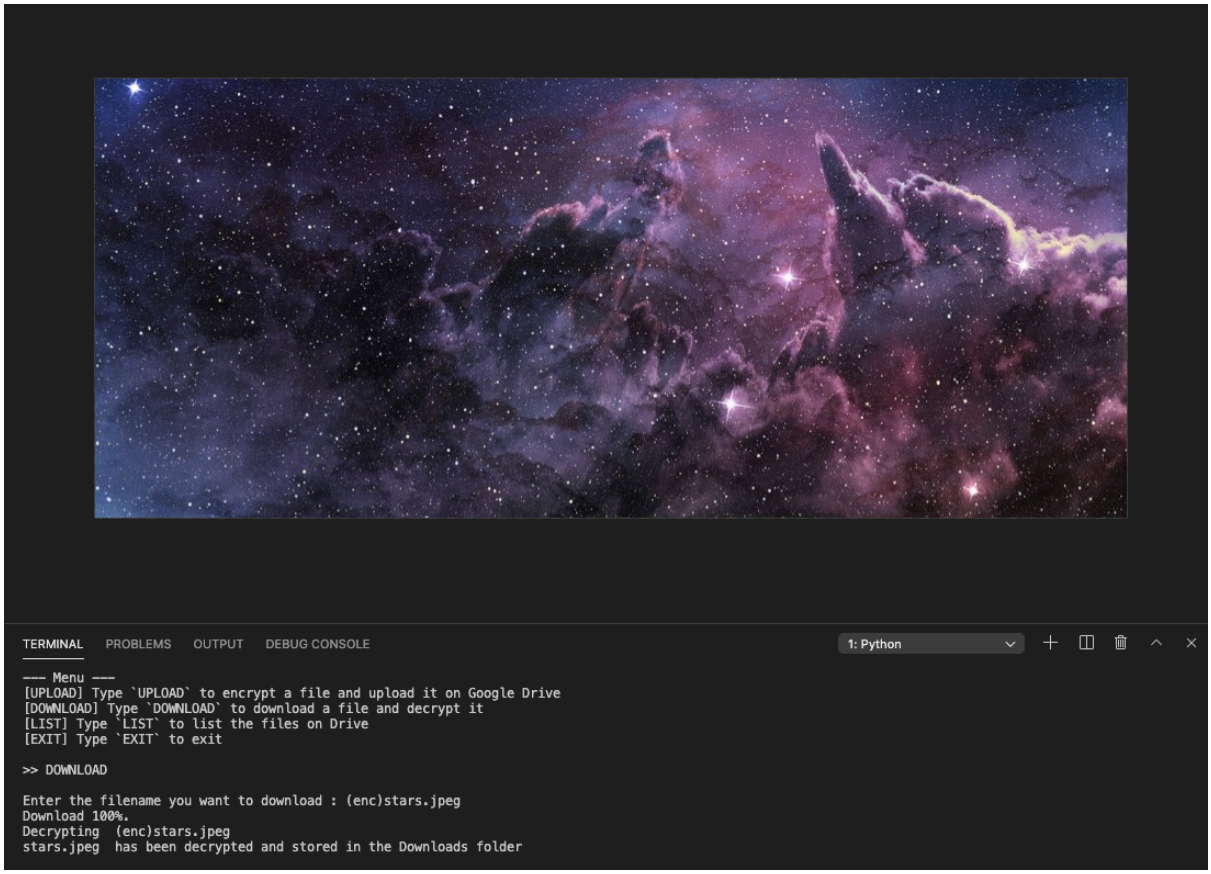
1. To upload files on Google Drive, the user selects the UPLOAD option.
2. Then the user selects a file from the Uploads folder to be uploaded.
3. The file is then encrypted and then Uploaded on Google Drive

```
WELCOME TO TANMAY's SECURE CLOUD APP
Please enter your CREDENTIALS to sign in


Username : Tanmay
Password:
Sign-In Successful

--- Menu ---
[UPLOAD] Type `UPLOAD` to encrypt a file and upload it on Google Drive
[DOWNLOAD] Type `DOWNLOAD` to download a file and decrypt it
[LIST] Type `LIST` to list the files on Drive
[EXIT] Type `EXIT` to exit

>> UPLOAD

Enter the filename you want to upload : stars.jpeg
File Uploaded on Google Drive
```

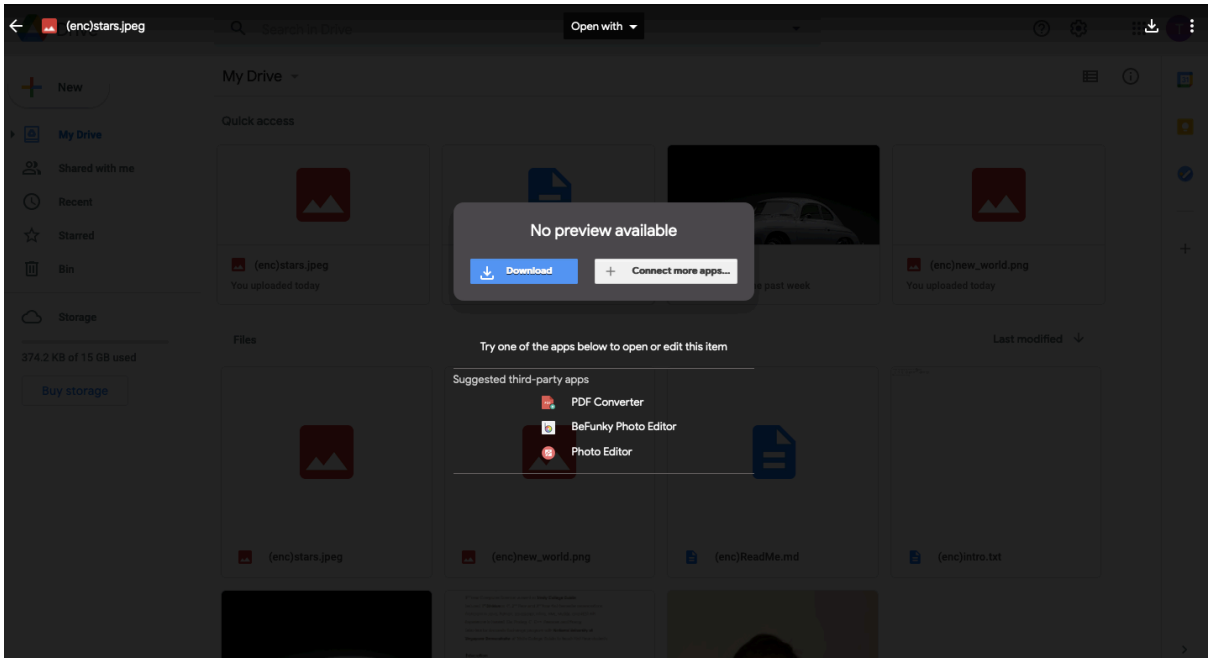


Download Files from Google Drive

1. To download files on Google Drive, the user selects the DOWNLOAD option.
2. Then the user selects a file from the Google Drive to be downloaded.
3. The file is then downloaded and then decrypted using the key.



However, the file on Google Drive is encrypted :



Verify that the Public Key and Private Keys are Correct

1. To verify the Public and Private keys generated, run “rsa_encryption.py” file.
2. When the user runs “rsa_encryption.py” file, the user will be prompted for a username.
3. Provide an username to check the Public and Private keys for that user. In the example below I verify the Public and Private key of “RUCHI” is correct.
4. To verify, I encrypt the AES key with Ruchi’s Public key and then decrypt it using Ruchi’s Private key. The AES key after encryption and decryption must be equal to the original AES key.

CODE LISTING

For easy instructions to run and test code, kindly access my GitHub page :

https://github.com/taaanmay/Secure-Cloud.git

(Link will be available after the submission date)

main.py

```
import os
import io
import sys
import GDrive
import admin


# Global Variable used to check if signed-in user is Admin or
Not
is_current_user_admin = False

def main():
    admin.sign_in()


if __name__ == '__main__':
    print("\n\nWELCOME TO TANMAY's SECURE CLOUD APP")
    main()
```

admin.py

```
import os
import io
import sys
import getpass
import pickle
import menus
import main
menu = menus
import rsa_encryption

rsa = rsa_encryption

# Function to write data in a file
def writeToFile(obj, filename):
    with open(filename, 'wb') as f:
        pickle.dump(obj, f, pickle.HIGHEST_PROTOCOL)

# Function to read data from the file
    # Stores Usernames and Passwords in a file called
accounts.pkl
def readFromFile(filename):
    if os.path.getsize("accounts.pkl") > 0:
        with open(filename, 'rb') as f:
            return pickle.load(f)
    else:
        return {}

# Loads all the usernames and passwords into accounts
accounts = readFromFile("accounts.pkl")

# Function to get user credentials to Sign-In
    # If there are no users, prompt the user to create a new
user
    # If there are users, prompt the user for their username
and password
def sign_in():
    try:
```

```
        # If no users in the application file, create a new
user
        if os.path.getsize("accounts.pkl") == 0:
            sign_up()

        # Get Username and Password for Authentication
        username, password = get_credentials()

        # If credentials match, show menu
            # If username is admin
            # If a regular group member, show regular menu
        if username in accounts and accounts[username] ==
password :
            if username == "admin" :
                print("Welcome, Admin! Sign-In Successful")

                # Global variable is_current_user_admin
                main.is_current_user_admin = True
                menu.get_admin_menu()

            else:
                print("Sign-In Successful")
                menu.get_regular_menu()

        # Credentials do not match. Ask if you want to try
again.
        else:
            print("Sign-In Unsuccessful. Do you want to try
again? (Yes or No)")
            response = str.upper(input())
            if response == "YES":
                sign_in()
            else :
                exit()

    except KeyboardInterrupt:
        exit()
```

```
# Function to retreive username and password from the user
input
def get_credentials():
    print("Please enter your CREDENTIALS to sign in\n\n")

    try :
        # Ask for Username
        username = str.lower(input("Username : "))

        # Get Pass method to get password
        password = getpass.getpass()

        return username, password
    except KeyboardInterrupt:
        exit()


# Function to sign-up a new user
def sign_up():

    try :
        print("--- SIGN-UP ---")
        print("\nPlease choose an Username & Password to Sign-
Up\n")
        if os.path.getsize("accounts.pkl") != 0:
            print("To go back, enter `BACK`")

        # Ask for Username
        username = str.lower(input("Username : "))

        # If user types BACK, go back to Admin Menu
        if str.upper(username) == "BACK" and
os.path.getsize("accounts.pkl") != 0:
            print("Going Back to Admin Menu...")

        else:
            duplicate_username = True
            # Check for no duplicate username
            while duplicate_username == True:
```

```
                if username in accounts or username == "back":
                    print("This username has been taken.
Please Choose a new one.")
                    # Ask for Username
                    username = str.lower(input("Username : "))
                else :
                    duplicate_username = False

                    # Get Pass method to get password
                    password = getpass.getpass()

                    # Store the username and password in
accounts
                    accounts[username] = password
                    writeToFile(accounts,"accounts.pkl")
                    print(".\n.\nSign-Up Successful.\n")

                    # Generate New User's Public and Private
Keys
                    print("Generating Public and Private Keys
for ",username)
                    status = rsa.generate_RSA_keys(username)
                    if status == True:
                        print("Public and Private Keys of the
user created in the Users folder")
                    else:
                        print("Public & Private keys NOT
generated. Please generate them for the user.")

                    # Navigate to Admin's Menu as only admin
can sign new people up
                    menu.get_admin_menu()

    except KeyboardInterrupt:
        exit()

def remover_User():
    print("-- REMOVE --")
```

```
    print("Please enter the username of the person to remove
the user from the group")
    print("To go back, enter `BACK`")
    try :
        # Ask for Username
        username = str.lower(input("Username : "))

        if str.upper(username) == "BACK":
            print("Going Back to Admin Menu...")

        # Check if Username is in accounts
        elif username in accounts:
            del accounts[username]
            writeToFile(accounts,"accounts.pkl")
            print(".\n.\n",username," Removed\n" )

        else:
            print(username," does not exist. Try Again or
enter `BACK`")
            remover_User()

    except KeyboardInterrupt:
        exit()

# Function to close the application
    # This function is used by different files as well
def exit():
    print("\n-->  TK Secure Cloud App Shutting Down...
\nGoodbye.")
    sys.exit()
```

auth.py

```
from __future__ import print_function
import os

import httplib2
```

```
from googleapiclient import discovery
from oauth2client import tools
from oauth2client.file import Storage
from oauth2client import client


try:
    import argparse
    flags =
argparse.ArgumentParser(parents=[tools.argparser]).parse_args(
)
except ImportError:
    flags = None

class auth:

    def
__init__(self,SCOPES,CLIENT_SECRET_FILE,APPLICATION_NAME):
        self.SCOPES = SCOPES
        self.CLIENT_SECRET_FILE = CLIENT_SECRET_FILE
        self.APPLICATION_NAME = APPLICATION_NAME



    def getCredentials(self):

        cwd_dir = os.getcwd()
        credential_dir = os.path.join(cwd_dir, '.credential')

        if not os.path.exists(credential_dir):
            os.makedirs(credential_dir)

        credential_path = os.path.join(credential_dir,
'google-drive-credentials.json')

        store = Storage(credential_path)
        credentials = store.get()
        if not credentials or credentials.invalid:
```

```
            flow =
client.flow_from_clientsecrets(self.CLIENT_SECRET_FILE,
self.SCOPES)
            flow.user_agent = self.APPLICATION_NAME
            if flags:
                credentials = tools.run_flow(flow, store,
flags)
            # else: # Needed only for compatibility with
Python 2.6
            #     credentials = tools.run(flow, store)
            print('Storing credentials to ' + credential_path)
        return credentials
```

encrypt.py

```
import os
import io
import sys


from Crypto import Random
import struct

from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

# Generate Key using Crypto.Random method and write in key.key
file
def generate_key():
    key = get_random_bytes(16)

    # Store key in key.key file
    file = open('key.key', 'wb')
    file.write(key)
    file.close()

# Retreive key from key.key file
def get_key():
    if os.path.getsize('key.key') != 0:
        file = open('key.key', 'rb')
        key = file.read()
        file.close()
        return key
    else:
        generate_key()
        return get_key()


# Encrypt file using key and return the encrypted file name
def encrypt_file(key, filename):
    path = "Uploads/"+filename
```

```
    #encrypted_file_name = filename + '.enc'
    encrypted_file_name = '(enc)'+filename
    #encrypted_file_path = 'Uploads/'+encrypted_file_name

    filesize = str(os.path.getsize(path)).zfill(16)
    chunk_size = 64*1024

    # Intialisation Vector = 16 random bytes
    iv = Random.new().read(16)

    encryptor = AES.new(key, AES.MODE_CBC, iv)

    with open(path,'rb') as infile:
        with open(encrypted_file_name, 'wb') as outfile:
            outfile.write(filesize.encode('utf-8'))
            outfile.write(iv)

            while True:
                chunk = infile.read(chunk_size)
                if len(chunk) == 0:
                    break
                elif len(chunk) % 16 != 0:
                    chunk += b' ' * (16 - (len(chunk) % 16))

                outfile.write(encryptor.encrypt(chunk))

    return encrypted_file_name


def decrypt_file(filename):

    key = get_key()
    path = "Downloads/" + filename

    print("Decrypting ", filename)

    chunk_size = 64*1024
```

```
    # Removing (enc) from  the filename
    decrypted_file_name = filename[5:]

    # Decryption
        # Open Encrypted File
    with open(path, 'rb') as infile:
        filesize = int(infile.read(16))
        iv = infile.read(16)
        decryptor = AES.new(key, AES.MODE_CBC, iv)

        with open("Downloads/"+decrypted_file_name, 'wb') as
outfile:
            while True:
                chunk = infile.read(chunk_size)
                if len(chunk) == 0:
                    break
                outfile.write(decryptor.decrypt(chunk))

            outfile.truncate(filesize)
    print(decrypted_file_name," has been decrypted and stored
in the Downloads folder")
    return decrypted_file_name
```

GDrive.py

```
# pylint: disable=unused-variable
import os
import io
import sys
import httplib2

from googleapiclient import discovery
from oauth2client import client
from oauth2client import tools
from oauth2client.file import Storage
from googleapiclient.http import MediaFileUpload,
MediaIoBaseDownload

import auth


APPLICATION_NAME = 'Drive Encryption'

SCOPES = ['https://www.googleapis.com/auth/drive']
CLIENT_SECRET_FILE = "/Users/tanmaykaushik/Desktop/TK-Cloud-
Secure/client_secret.json"

authInst =
auth.auth(SCOPES,CLIENT_SECRET_FILE,APPLICATION_NAME)
credentials = authInst.getCredentials()

http = credentials.authorize(httplib2.Http())
service = discovery.build('drive', 'v3', http=http)

# Function to list 10 files present in the Google Drive
def list_files() :

    # Call the Drive v3 API
    # pylint: disable=maybe-no-member
    results = service.files().list(
        pageSize= 10, fields="nextPageToken, files(id,
name)").execute()
```

```
    items = results.get('files', [])

    if not items:
        print('No files found.')
    else:
        print('\n\nFiles on Google Drive: \n')
        count = 1
        for item in items:
            print(u'({0}){1}'.format(count,item['name']))
            count = count + 1

    print("\n")

# Function that returns the file_ID of the required file
def get_file_id(request):
    file_ID = -1

    # pylint: disable=maybe-no-member
    results = service.files().list(
    pageSize = 1, fields="nextPageToken, files(id, name, kind,
mimeType)",q=request).execute()
    items = results.get('files', [])

    # Check if File Found
        # If yes, return file_ID else -1
    if not items:
        file_ID = -1
    else:
        for item in items:
            file_ID = item['id']
    return file_ID

# Function to download file from the Google Drive
def download_file(file_ID, filename) :

    # pylint: disable=maybe-no-member
    request = service.files().get_media(fileId=file_ID)
    fh = io.BytesIO()
    downloader = MediaIoBaseDownload(fh, request)
```

```
    done = False
    while done is False:
        status, done = downloader.next_chunk()
        print("Download %d%%." % int(status.progress() * 100))

    # Store the File in the Downloads Folder
    with io.open("Downloads/" + filename,'wb') as f:
        fh.seek(0)
        f.write(fh.read())


# Function to Upload file to Google Drive
def upload_file(filename, path):

    mimetype = file_type(filename)

    file_metadata = {'name': filename}
    media = MediaFileUpload(path, mimetype=mimetype)

    # pylint: disable=maybe-no-member
    file = service.files().create(body=file_metadata,
                                  media_body=media,
                                  fields='id').execute()

    print("File Uploaded on Google Drive")

# Function to get the file extenstion
def file_type(filename):

    mime_types = dict(
        txt='text/plain',
        htm='text/html',
        html='text/html',
        php='text/html',
        css='text/css',
        js='application/javascript',
        json='application/json',
        xml='application/xml',
        swf='application/x-shockwave-flash',
```

```
        flv='video/x-flv',

        # images
        png='image/png',
        jpe='image/jpeg',
        jpeg='image/jpeg',
        jpg='image/jpeg',
        gif='image/gif',
        bmp='image/bmp',
        ico='image/vnd.microsoft.icon',
        tiff='image/tiff',
        tif='image/tiff',
        svg='image/svg+xml',
        svgz='image/svg+xml',

        # archives
        zip='application/zip',
        rar='application/x-rar-compressed',
        exe='application/x-msdownload',
        msi='application/x-msdownload',
        cab='application/vnd.ms-cab-compressed',

        # audio/video
        mp3='audio/mpeg',
        ogg='audio/ogg',
        qt='video/quicktime',
        mov='video/quicktime',

        # adobe
        pdf='application/pdf',
        psd='image/vnd.adobe.photoshop',
        ai='application/postscript',
        eps='application/postscript',
        ps='application/postscript',

        # ms office
        doc='application/msword',
        rtf='application/rtf',
        xls='application/vnd.ms-excel',
```

```
        ppt='application/vnd.ms-powerpoint',

        # open office
        odt='application/vnd.oasis.opendocument.text',
        ods='application/vnd.oasis.opendocument.spreadsheet',
    )

    file_extenstion = os.path.splitext(filename)[1]
[1:].lower()
    if file_extenstion in mime_types:
        return mime_types[file_extenstion]
    else:
        return 'application/octet-stream'
```

menus.py

```
import os
import io
import sys

import GDrive
import admin
import main
import encrypt
import rsa_encryption

googleDrive = GDrive
encryption = encrypt

# Function to print options available to Admin
def print_admin_options():
    print('[ADD] Type `ADD` to add a new user to group')
    print('[DEL] Type `DEL` to delete a user from the group')
    print('[MENU] Type `MENU` to go to the files menu')
    print('[EXIT] Type `EXIT` to exit')

# Function to give admin-menu to admin and take input
    # 4 options - ADD, Remove, Menu, Exit
def get_admin_menu():
    exit = False
    while exit == False:
        try:
            print("--- Admin Menu ---")

            # Print the list of options available to Admin
            print_admin_options()

            # Take Input and check what the admin wants to do
                # If Input is not recognised, give the options
again
            user_input = input("\n>> ")

            if str.upper(user_input) == "ADD":
```

```
                admin.sign_up()

            elif str.upper(user_input) == "DEL":
                admin.remover_User()

            elif str.upper(user_input) == "MENU":
                get_regular_menu()

            elif str.upper(user_input) == "EXIT":
                exit = True

            else:
                print("Input not recognised. Please select an
option again")
                get_admin_menu()

        except KeyboardInterrupt:
            admin.exit()

    admin.exit()


# Function to print options available to the group
    # If admin is signed-in, an extra feature given to go to
the Admin's Menu
def print_regular_menu_options():
    print('[UPLOAD] Type `UPLOAD` to encrypt a file and upload
it on Google Drive')
    print('[DOWNLOAD] Type `DOWNLOAD` to download a file and
decrypt it')
    print('[LIST] Type `LIST` to list the files on Drive')
    if main.is_current_user_admin == True:
        print('[ADMIN] Type `ADMIN` to access Admin Menu')
    print('[EXIT] Type `EXIT` to exit')
    #print('[MENU] Type `MENU` to go to the files menu')


# # Function to give menu to goup member and take input
```

```
    # 3 options - UPLOAD, DOWNLOAD, LIST, EXIT
    # If user is admin, 1 option more - Go to ADMIN menu
def get_regular_menu():
    exit = False
    while exit == False:
        try:
            print("\n--- Menu ---")

            # Print the list of options available to Admin
            print_regular_menu_options()

            # Take Input and check what the admin wants to do
                # If Input is not recognised, give the options
again
            user_input = input("\n>> ")

            if str.upper(user_input) == "UPLOAD":
                filename = input("\nEnter the filename you
want to upload : ")
                path = "Uploads/" + filename

                if os.path.exists(path):
                    # Encrypt the file
                    encrypted_file_name =
encryption.encrypt_file(encryption.get_key(), filename)

                    # If file was encrypted upload the file
                    if encrypted_file_name != None:
                        # Update Path to the Encrypted File
                        #new_path = "Uploads/" +
encrypted_file_name
                        new_path = encrypted_file_name
                        # Upload File

googleDrive.upload_file(encrypted_file_name, new_path)

                        # Remove encrypted file
                        os.remove(encrypted_file_name)
```

```
                    # If file was not encrypted, ask the user
if wants to upload the file without Encryption
                    else:
                        resp = input("File could not be
encripted. Do you want to upload the file without encryption?
(Yes/No)")
                        if str.upper(resp) == "YES":
                            googleDrive.upload_file(filename,
path)
                        else:
                            print("File not uploaded.")

                else:
                    print("PATH >> ",path)
                    print("File could not be found. Please
check if the file is in Uploads Folder")

            elif str.upper(user_input) == "DOWNLOAD":

                filename = input("\nEnter the filename you
want to download : ")
                # Get File ID
                file_ID = googleDrive.get_file_id("name
contains '"+filename+"'")

                if file_ID != -1 :
                    # Download File from the drive
                    googleDrive.download_file(file_ID,
filename)

                    # Decrypt the File and Store in Download
Folder
                    down_file =
encryption.decrypt_file(filename)

                    # Delete the downloaded encrypted file
                    os.remove("Downloads/"+filename)
                else:
```

```
                    print("File Not Found on Drive.")

            elif str.upper(user_input) == "LIST":
                googleDrive.list_files()

            elif str.upper(user_input) == "ADMIN" and
main.is_current_user_admin == True:
                get_admin_menu()

            elif str.upper(user_input) == "EXIT":
                exit = True

            else:
                print("Input not recognised. Please select an
option again")
                get_regular_menu()

        except KeyboardInterrupt:
            admin.exit()

    admin.exit()
```

rsa_encryption.py

```
import io
import os
import rsa
from cryptography.fernet import Fernet

# This program is used to produce Public and Private Keys.
# Key.key (which is used by AES algorithm) file is the text we
want to encrypt using Public and Private Keys.
# Reason - AES is the key which can decrypt any file on the
Google Drive
# Solution - We use RSA protocol to share key.key file using
Public and Private Keys


# key.key = Plain Text

# Function to Generate Public and Private Keys
    # Takes Username and Creates a folder in the User's
Section with that name
    # Generates Public and Private Keys for that user
    # Stores the User's Public and Private Keys in the user's
folder
def generate_RSA_keys(username):
    try:
        # Create a Folder in the Users folder with the name of
the new User
        directory = username
        parent_dir = "Users/"
        path = os.path.join(parent_dir, directory)
        os.mkdir(path)

        # Generate Public Key and Private Key for the user
        (public_key, private_key) = rsa.newkeys(2048)

        # Write Public Key in file called public.key
        pub_path = path+"/public.key"
        public_file = open(pub_path, 'wb')
```

```
        public_file.write(public_key.save_pkcs1('PEM'))
        public_file.close()

        # Write Private Key in file called private.key
        priv_path = path+"/private.key"
        private_file = open(priv_path, 'wb')
        private_file.write(private_key.save_pkcs1('PEM'))
        private_file.close()

        status = True
        return status

    except:
        status = False
        return status

# Function to encrypt AES key using Public Key of the selected
User
def rsa_encrypt(username):
    try:
        # Get Data from the key.key (AES Key) which is Plain
Text
        aes_key = open('key.key', 'rb')
        key_data = aes_key.read()

        # Encrypt the key.key file using Public Key of the
user
        # Get the key from the public key file
        path = 'Users/'+username
        temp_path = path +'/public.key'
        pub_key = open(temp_path,'r')
        pub_key_data = pub_key.read()

        # Load the Data from the file
        public_key = rsa.PublicKey.load_pkcs1(pub_key_data)

        # Encrypt the AES key
        encrypted_aes_key = rsa.encrypt(key_data, public_key)
```

```
        # Create a file called AES_encrypted
        new_path = path+'/AES_encrypted'
        file = open(new_path,'wb')
        file.write(encrypted_aes_key)
        print("AES key successfully encrypted using Public
Key")
        print("Your encrypted AES key is this :
\n\n",encrypted_aes_key)
    except KeyboardInterrupt():
        print("AES key could not be encrypted using the Public
Key")

def rsa_decrypt(username):

    # Get the Private Key file
    path = 'Users/'+username
    temp_path = path +'/private.key'
    priv_key = open(temp_path,'rb')
    private_key_data = priv_key.read()

    # Get Private Key
    private_key = rsa.PrivateKey.load_pkcs1(private_key_data)

    # Get Encrypted AES File
    new_path = path+'/AES_encrypted'
    e = open(new_path,'rb')
    aes_enc = e.read()

    decrypted_key = rsa.decrypt(aes_enc, private_key)

    print("\n\nAES Key after RSA and Decryption Encryption  =
", decrypted_key)
    print("\n")


def main():
```

```
    username = input("\n\nYou can test RSA encryption by
running this file. Type a name of any user from the User's
Folder.\n>>")
    # Check Encryption
        # See if a new file by the name of AES_encrypted is
created in the User's Folder
    # Encrypt and Decrypt
    rsa_encrypt(username)
    rsa_decrypt(username)

    # Orignal AES Key
    aes_key = open('key.key', 'rb')
    key_data = aes_key.read()
    print("\nORIGNAL AES Key = ",key_data)


if __name__ == '__main__':
    main()
```